Introduction to System Modelling and Control Design Using Matlab/Simulink

1. Introduction

This Virtual Laboratory involves exercises using Matlab and Simulink to perform modelling and simulation of dynamic systems.

You will follow a series of steps that will guide you to build relevant models and analyse system performance.

The submission of this coursework will be done via Blackboard. You will be asked to answer various questions and in some cases to upload the graphical output you generated.

Matlab coding procedure

- The most convenient way is to write code as .m format script file and execute it.
- Alternatively you may find it more convenient to use .mlx file format (*live script*) which displays the output in the same window alongside the relevant code
- In both cases the file can be split into sections (*insert section break*).
- Each section can be run separately. Beware that once set, variable values will remain unchanged until the code to reassign values is executed

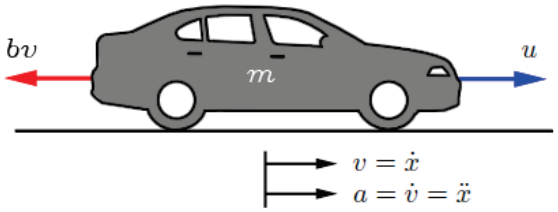
Steps to upload graphical output:

- Matlab figures can be saved as matlab figure files (*.fig), then uploaded on Blackboard as required
- Alternatively you can use ***Printscreen*** or equivalent command to save a selected window or a portion of your computer screen.
    - On Windows you can use key combination <Shift+Win+s> to invoke the *Snip & Sketch* tool, and select the relevant portion of the screen.
    - Save the image a JPG file, and then
    - Upload the file on Blackboard as required

The exercises are based on those available on http://ctms.engin.umich.edu/CTMS/ but the model parameters have been changed, so your results will not be the same as those.

2. Car Cruise Control System

Automatic **cruise control** is an excellent example of a feedback control system found in many modern vehicles. The purpose of the cruise control system is to maintain a constant vehicle speed despite external **disturbances**, such as changes in wind or road grade. This is accomplished by measuring the vehicle speed, comparing it to the desired or **reference** speed, and automatically adjusting the throttle according to a **control law**.



We consider here a simple model of the vehicle dynamics, shown in the free-body diagram (FBD) above. The vehicle, of mass m, is acted on by a control force, u. The force u represents the force generated at the road/tire interface. For this simplified model we will assume that we can control this force directly and will neglect the dynamics of the powertrain, tires, etc., that go into generating the force. The resistive forces, bv, due to rolling resistance and wind drag, are assumed to vary linearly with the vehicle velocity, v, and act in the direction opposite the vehicle's motion.

With these assumptions we are left with a first-order mass-damper system. Summing forces in the x-direction and applying Newton's 2nd law, we arrive at the following system equation:

$$m\dot{v} + bv = u$$

Since we are interested in controlling the speed of the vehicle, the output equation is chosen as follows

$$y = v$$

Performance specifications

The next step is to come up with some **design criteria** that the compensated system should achieve. When the engine gives a 500 Newton force, the car will reach a maximum velocity of 10 m/s (22 mph), see open-loop step response section below. An automobile should be able to accelerate up to that speed in less than 5 seconds. In this application, a 10% overshoot and 2% steady-state error on the velocity are sufficient.
Keeping the above in mind, we have proposed the following design criteria for this problem:

- Rise time < 5 s
- Overshoot < 10%
- Steady-state error < 2%

2.1 Matlab modelling

For this example, let's assume that the parameters of the system are:

```
(m)    vehicle mass           1000 kg
(b)    damping coefficient    50 N.s/m
(u)    nominal control force   500 N
(r)    reference (desired)speed          10 m/s
```

Taking the Laplace transform of the governing differential equation and assuming zero initial conditions, we find the transfer function of the cruise control system to be:

$$P(s) = \frac{V(s)}{U(s)} = \frac{1}{ms + b} \qquad [\frac{m/s}{N}]$$

We enter the transfer function model into MATLAB using the following commands:

```
s = tf('s');
P_cruise = 1/(m*s+b);
```

2.2 Open-loop step response

The **open-loop** response of the system, without any feedback control, to a step input force of 500 Newtons is simulated in MATLAB as follows:

```
m = 1000;
b = 50;
u = 500;

s = tf('s');
P_cruise = 1/(m*s+b);

step(u*P_cruise)
```

Questions:

Q 1 Upload open loop step response plot on **Blackboard**

For the open-loop system, estimate the following:

Q 2(a) Steady state speed =

Q 2(b) Rise time =

Q 3 Is the design specification satisfied?

You should see that the open-loop system exhibits no overshoot or oscillations (characteristic of first-order systems), and does reach the desired steady-state speed of 10 m/s; however, the rise time is much too slow, ~60 s. Therefore we need to design a feedback controller which speeds up the response significantly without negatively affecting the other dynamic performance metrics.

2.3 Open-loop poles/zeros

The cruise control system has a single pole. This can be evaluated using the command

```
pole(P_cruise)
```

and also plotted on the s-plane using the following MATLAB commands:

```
pzmap(P_cruise)
axis([-1 1 -1 1])
```

Questions:

Q 4 Upload Pole-Zero Map on Blackboard

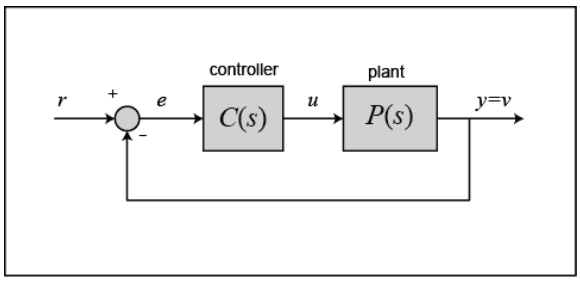
Q 5 What is the location of the open loop pole?

Q 6 Is the open loop system stable?

We observe that the **open-loop** system is stable and does not oscillate since the pole is real and negative. Furthermore, the speed of response is determined by the magnitude of this pole, `b/m`: the larger the magnitude, the quicker the system approaches the steady-state value. Since we're typically not able to change the system parameters to change the dynamic response of the system, we must instead design controllers which alter the poles and zeros of the **closed-loop** system to meet the desired performance specifications.

2.4 Closed loop control

The block diagram of a typical unity feedback system is shown below.



The transfer function of a PID controller is

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

We can define a PID controller in MATLAB using the transfer function directly:

```
Kp = 1;
Ki = 1;
Kd = 1;

s = tf('s');
C = Kp + Ki/s + Kd*s
```

Alternatively, we may use MATLAB's **pid controller object** to generate an equivalent continuous time controller as follows:

```
C = pid(Kp,Ki,Kd)
```

```
C =
                1
  Kp + Ki * --- + Kd * s
                s
  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
```

2.5 Proportional control

The first thing to do in this problem is to find a closed-loop transfer function with a proportional control ($C = K_p$) added.

By reducing the unity feedback block diagram, the closed-loop transfer function with a proportional controller becomes:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_p}{ms + b + K_p}$$

A proportional controller, K_p, decreases the rise time, which is desirable in this case.

For now, use K_p equal to 100 and a reference speed of 10 m/s. Create a new $\mathrm{m\text{-}file}$ and enter the following commands.

```
m = 1000;
b = 50;
r = 10;

s = tf('s');
P_cruise = 1/(m*s + b);

Kp = 100;
C = pid(Kp);

T = feedback(C*P_cruise,1)

t = 0:0.1:20;
step(r*T,t)
axis([0 20 0 10])
```

Questions:

For K_p=100:

Q 7 Upload the closed loop step response on Blackboard

Q 8 What is the location of the closed loop pole?

Q 9 What is the time constant in seconds of the closed loop system?

Q 10 Is the closed loop system stable?

Q 11 What is the steady state speed error of the closed loop system?

Q 12 Does the steady state error satisfy design criteria?

Q 13 Does the rise time satisfy design criteria?

Note that we have used the MATLAB feedback command to simplify the block diagram reduction of the closed-loop system. Please verify for yourself that the result agrees with the closed-loop transfer function, T, derived above.

Running the m-file in MATLAB should give you the step response above. As you can see from the plot, neither the steady-state error nor the rise time satisfy our design criteria.

You can increase the proportional gain, K_p, to reduce the rise time and the steady-state error. Change the existing m-file so that K_p equals 5000 and rerun it in the MATLAB command window.

```
Kp = 5000;
C = pid(Kp);
T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```

Questions:

For K_p=5000:

Q 14 Upload the closed loop step response on Blackboard

Q 15 What is the location of the closed loop pole:

Q 16 What is the time constant in seconds of the closed loop system?

Q 17 Is the closed loop system stable?

Q 18 What is the steady state speed error of the closed loop system?

Q 19 Does the steady state error satisfy design criteria?

Q 20 Does the rise time satisfy design criteria?

Both the steady-state error and the rise time have been reduced substantially. However, this response is unrealistic because a real cruise control system generally cannot change the speed of the vehicle from 0 to 10 m/s too quickly due to power limitations of the engine and the drivetrain.

Actuator limitations are very frequently encountered in practice in control systems engineering, and consequently, the required control action must always be considered when proposing a new controller.

The solution to this problem in this case is to choose a lower proportional gain, K_p, that will give a reasonable rise time, and add an integral controller to eliminate the steady-state error.

To estimate the actuator effort *u* during step response, we can redefine the closed loop system to make *u* the output signal.

```
Kp = 5000;
C = pid(Kp);
T = feedback(C,P_cruise)

step(r*T,t)
```

Questions:

Q 21 Upload the step response plot *u* vs. *t* on Blackboard

Q 22 What is the largest propulsion force during step response?

The solution to this problem in this case is to choose a lower proportional gain, K_p, that will give a reasonable rise time, and add an integral controller to eliminate the steady-state error.

Try lower K_p, values, to estimate the largest one satisfying the 500 N propulsion force limit.

Questions:

Q 23 What is the largest K_p that does will not exceed the 500 N propulsion force limit?

Q 24 For K_p in Q23, upload the actuator force step response plot *u* vs. *t* on Blackboard

Q 25 For K_p in Q23, upload the output step response plot *y* vs. *t* on Blackboard

Q 26 For K_p in Q23, what is the steady state speed error of the closed loop system?

2.6 PI control

The closed-loop transfer function of this cruise control system with a PI controller ($C = K_p + K_i/s$) is:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_p s + K_i}{ms^2 + (b + K_p)s + K_i}$$

Addition of an integral controller to the system eliminates the steady-state error. For now, let K_p equal 600 and K_i equal 1 and see what happens to the response. Change your m-file to the following.

```
Kp = 600;
Ki = 1;
C = pid(Kp,Ki);

T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```

Questions:

For K_p = 600, K_i = 1;

Q27 Upload the closed loop step response on Blackboard

Q28 What is the location of the closed loop poles?

Q29 What is the steady state speed error of the closed loop system?

Q30 What is the time constant in seconds of the closed loop system?

Now adjust both the proportional gain, K_p, and the integral gain, K_i, to obtain the desired response. When you adjust the integral gain, K_i, we suggest you to start with a small value since a large K_i can de-stabilize the response. When K_p equals 800 and K_i equals 40, the step response will look like the following:

```
Kp = 800;
Ki = 40;
C = pid(Kp,Ki);

T = feedback(C*P_cruise,1);

step(r*T,t)
axis([0 20 0 10])
```

Questions:

For K_p = 800, K_i = 40:

Q31 Upload the closed loop step response on Blackboard

Q32 What is the location of the closed loop poles?

Q33 What is the steady state speed error of the closed loop system?

Q34 What is the time constant in seconds of the closed loop system?

2.7 PID control

For this particular example, no implementation of a derivative controller was needed to obtain the required output. However, you might want to see how to work with a PID control for the future reference. The closed-loop transfer function for this cruise control system with a PID controller ($C = K_p + K_i/s + K_d s$) is:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)C(s)}{1 + P(s)C(s)} = \frac{K_d s^2 + K_p s + K_i}{(m + K_d)s^2 + (b + K_p)s + K_i} \quad (5)$$

Let K_p equal 1, K_i equal 1, and K_d equal 1 and enter the following commands into a new m-file.

```
Kp = 1;
Ki = 1;
Kd = 1;
C = pid(Kp,Ki,Kd);

T = feedback(C*P_cruise,1);
```

Plot the step response

Questions:

For K_p = 1, K_i = 1, K_d=1:

Q35 Upload the closed loop step response on Blackboard

Q36 What is the steady state speed error of the closed loop system?

Q37 What is the time constant in seconds of the closed loop system?

Now try to adjust all of K_p, K_d, and K_i until you obtain satisfactory results. We will leave this as an exercise for you to work on.

Suggestion: Usually choosing appropriate gains requires a trial and error process. The best way to attack this tedious process is to adjust one variable (K_p, K_i, or K_d) at a time and observe how changing one variable influences the system output.

3. Simulink: Modelling

In Simulink, it is very straightforward to represent and then simulate a mathematical model representing a physical system. Models are represented graphically in Simulink as block diagrams. A wide array of blocks are available to the user in provided libraries for representing various phenomena and models in a range of formats. One of the primary advantages of employing Simulink (and simulation in general) for the analysis of dynamic systems is that it allows us to quickly analyze the response of complicated systems that may be prohibitively difficult to analyze analytically. Simulink is able to numerically approximate the solutions to mathematical models that we are unable to, or don't wish to, solve "by hand."

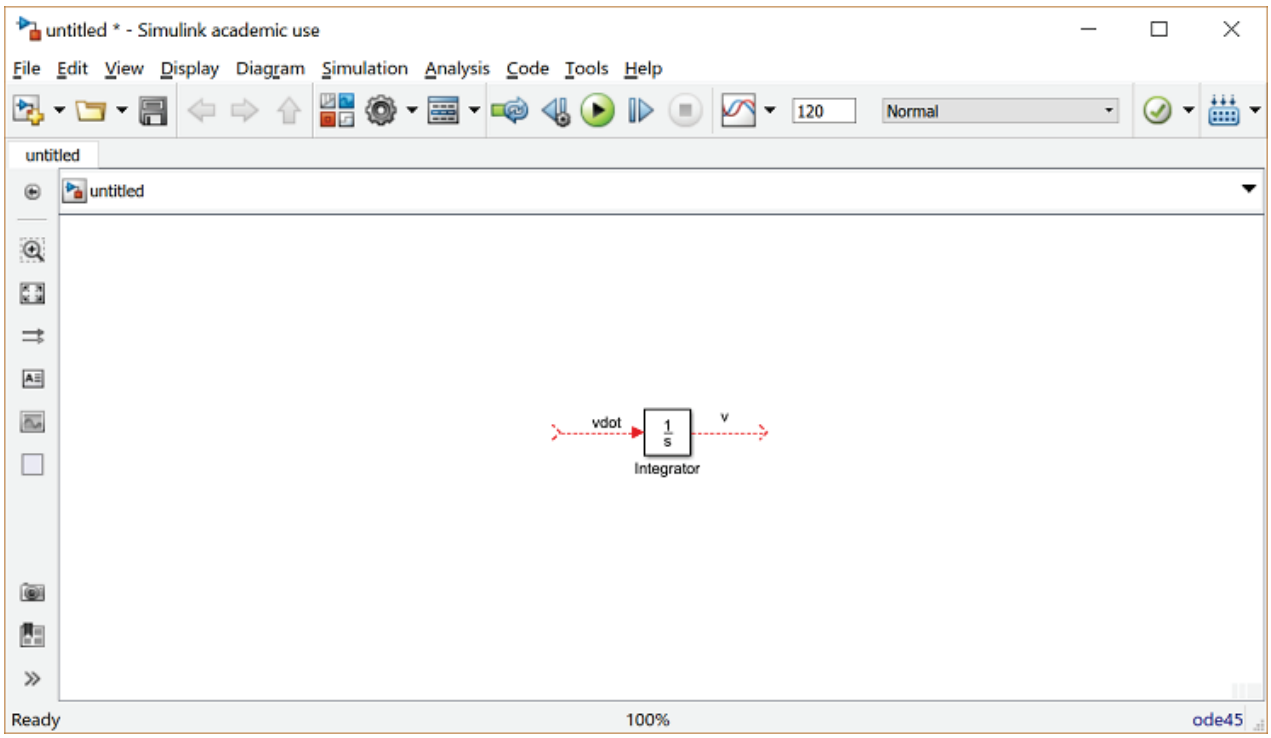
In general, the mathematical equations representing a given system that serve as the basis for a Simulink model can be derived from physical laws.

3.1 Building the model

This system will be modeled by summing the forces acting on the mass and integrating the acceleration to give the velocity. Open Simulink and open a new model window. First, we will model the integral of acceleration.

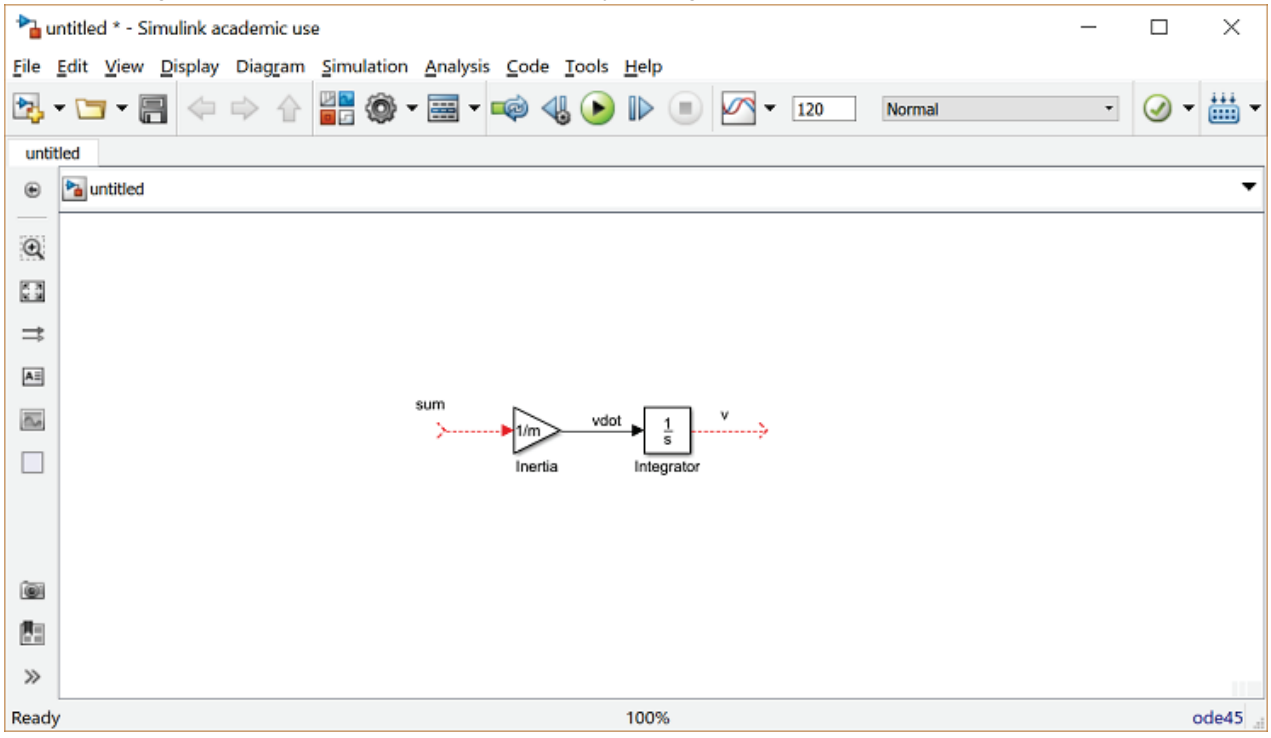
$$\int \frac{dv}{dt}\, dt = v$$

- Insert an Integrator block (from the Continuous library) and draw lines to and from its input and output terminals.
- Label the input line "vdot" and the output line "v" as shown below. To add such a label, double click in the empty space just above the line.
-



Since the acceleration (dv/dt) is equal to the sum of the forces divided by mass, we will divide the incoming signal by the mass.

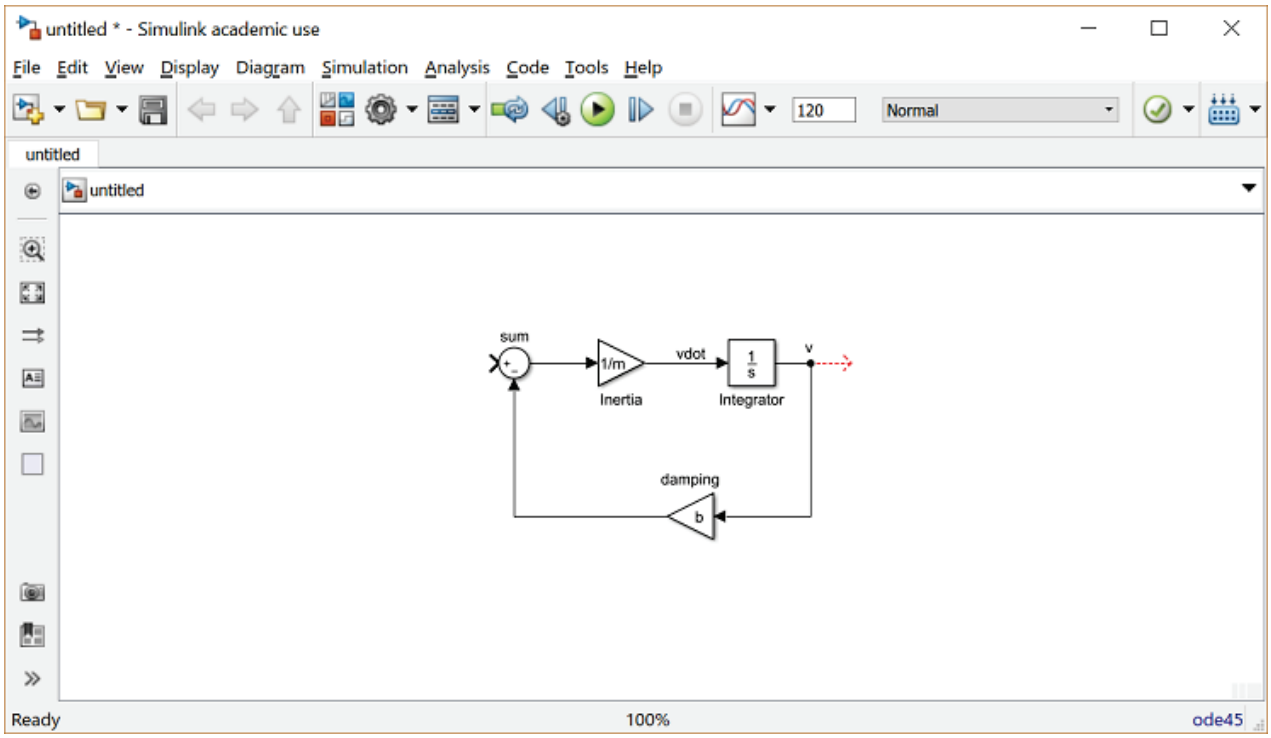
- Insert a Gain block (from the Math Operations library) connected to the Integrator block input line and draw a line leading to the input of the Gain block.
- Edit the Gain block by double-clicking on it and change its value to "1/m".
- Change the label of the Gain block to "inertia" by clicking on the word "Gain" underneath the block.



Now, we will add in the forces which are represented in Equation (1). First, we will add in the damping force.

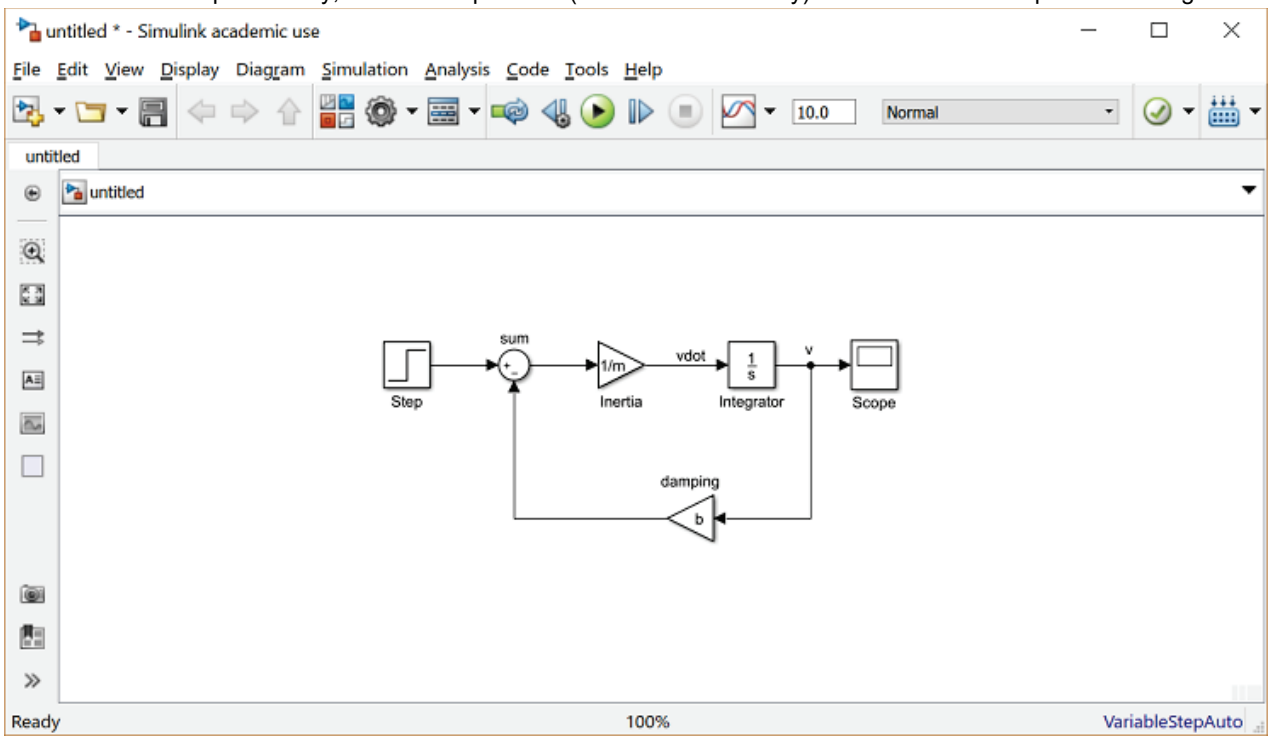
- Attach a Sum block (from the Math Operations library) to the line leading to the inertia Gain block.
- Change the signs of the Sum block to "+-".
- Insert a Gain block below the Inertia block, select it by single-clicking on it, and select **Flip Block** from the **Rotate & Flip** menu (or type **Ctrl-I**) to flip it left-to-right.
- Set the block's value to "b" and rename this block to "damping".

- Tap a line (hold **Ctrl** while drawing) off the Integrator block's output and connect it to the input of the damping Gain block.
- Draw a line from the damping Gain block output to the negative input of the Sum Block.

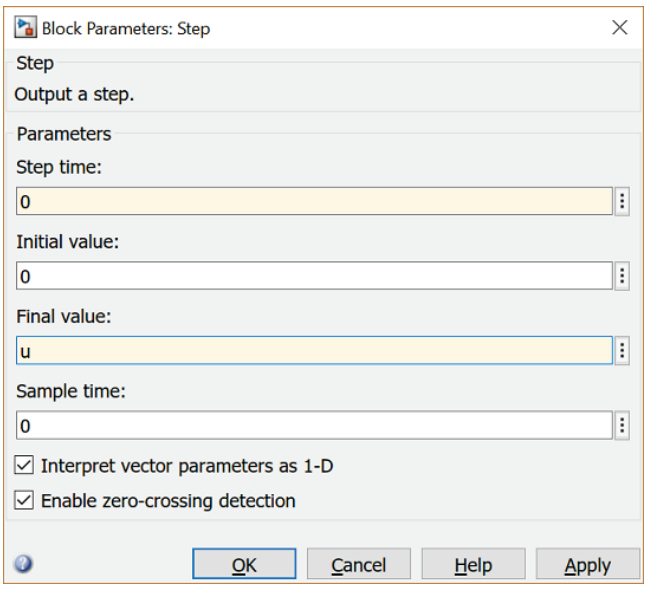


The second force acting on the mass is the control input, u. We will apply a step input.

- Insert a Step block (from the Sources library) and connect it with a line to the positive input of the Sum Block.
- To view the output velocity, insert a Scope block (from the Sinks library) connected to the output of the Integrator.



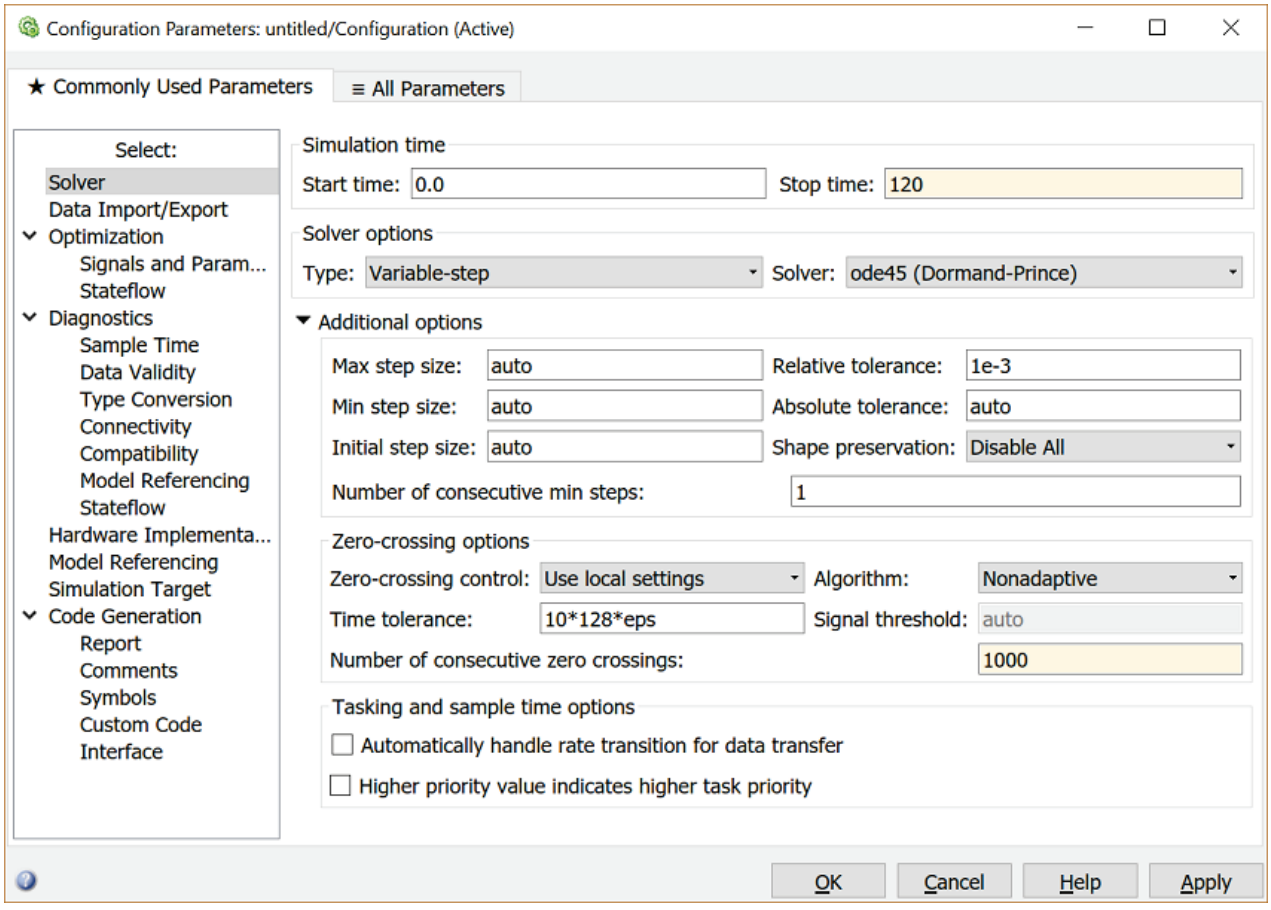
- To provide an appropriate step input of 500 at time equals zero, double-click the Step block and set the Step Time to "0" and the Final Value to "u".



3.2 Open-loop response

To simulate this system, first, an appropriate simulation time must be set.

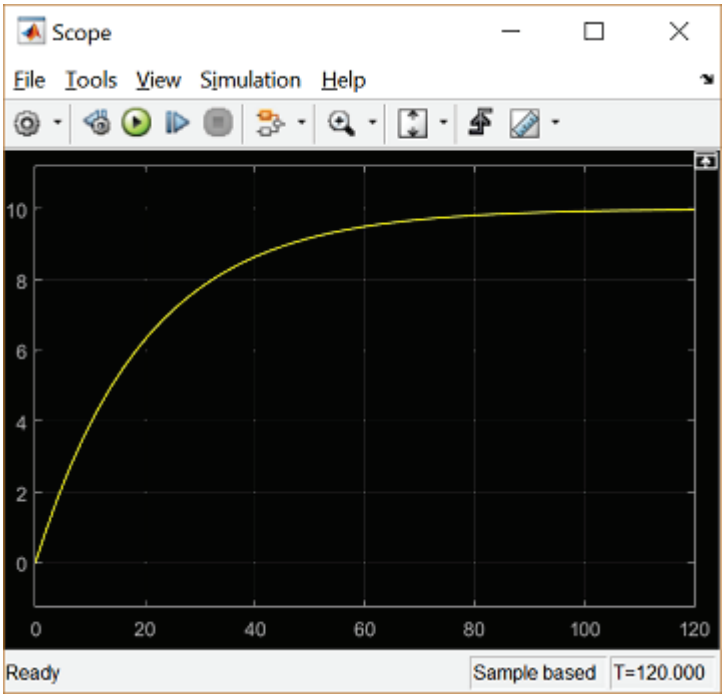
- Select Parameters from the Simulation menu and enter "120" in the Stop Time field. 120 seconds is long enough to view the open-loop response.



The physical parameters must now be set. Run the following commands at the MATLAB prompt:

```
m = 1000;
b = 50;
u = 500;
```

Run the simulation (hit **Ctrl-T** or select **Run** from the **Simulation** menu). When the simulation is finished you should see the following output.



Observing the above, we would like to improve the response of the cruise control system. The model created here will be employed for controller design and analysis within Simulink in the Cruise Control: Simulink Controller Design page.

3.3 Extracting a linear model into MATLAB

A linear model of the system (in state space or transfer function form) can be extracted from a Simulink model into MATLAB. This is done through the use of In1 and Out1 blocks and the MATLAB function `linmod`.

- Replace the Step Block and Scope Block with an In1 and an Out1 block, respectively (these blocks can be found in the Ports & Subsystems library). This defines the input and output of the system for the extraction process.

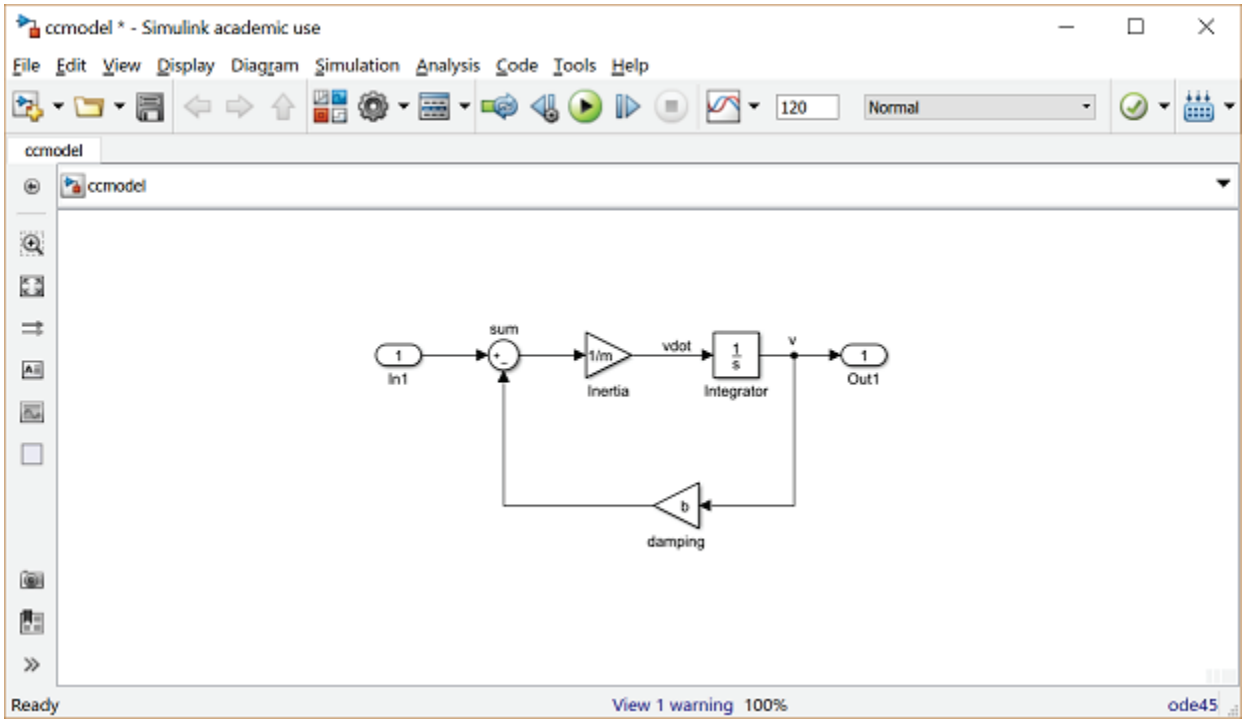


Figure 3.1: Simulink model of the car

Save your file as "ccmodel.slx" (select **Save As** from the **File** menu). MATLAB will extract the linear model from the saved model file, not from the open model window. At the MATLAB prompt, enter the following commands:

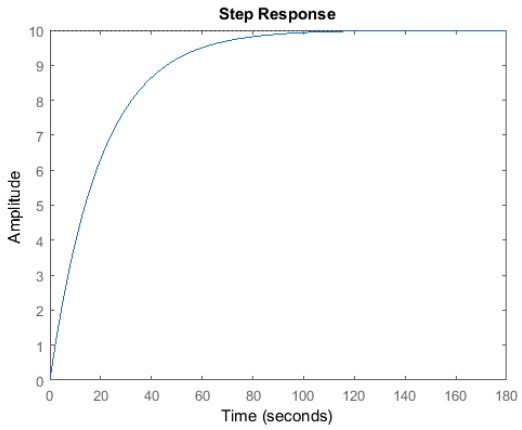
```
m = 1000;
b = 50;
u = 500;
[A,B,C,D] = linmod('ccmodel')
cruise_ss = ss(A,B,C,D);
```

```
A =
   -0.0500
B =
   1.0000e-03
C =
     1
D =
     0
```

To verify the model extraction, we will generate an open-loop step response of the extracted transfer function in MATLAB. We will multiply the numerator by 500 to simulate a step input of 500 N. Enter the following command in MATLAB.

```
step(u*cruise_ss)
```

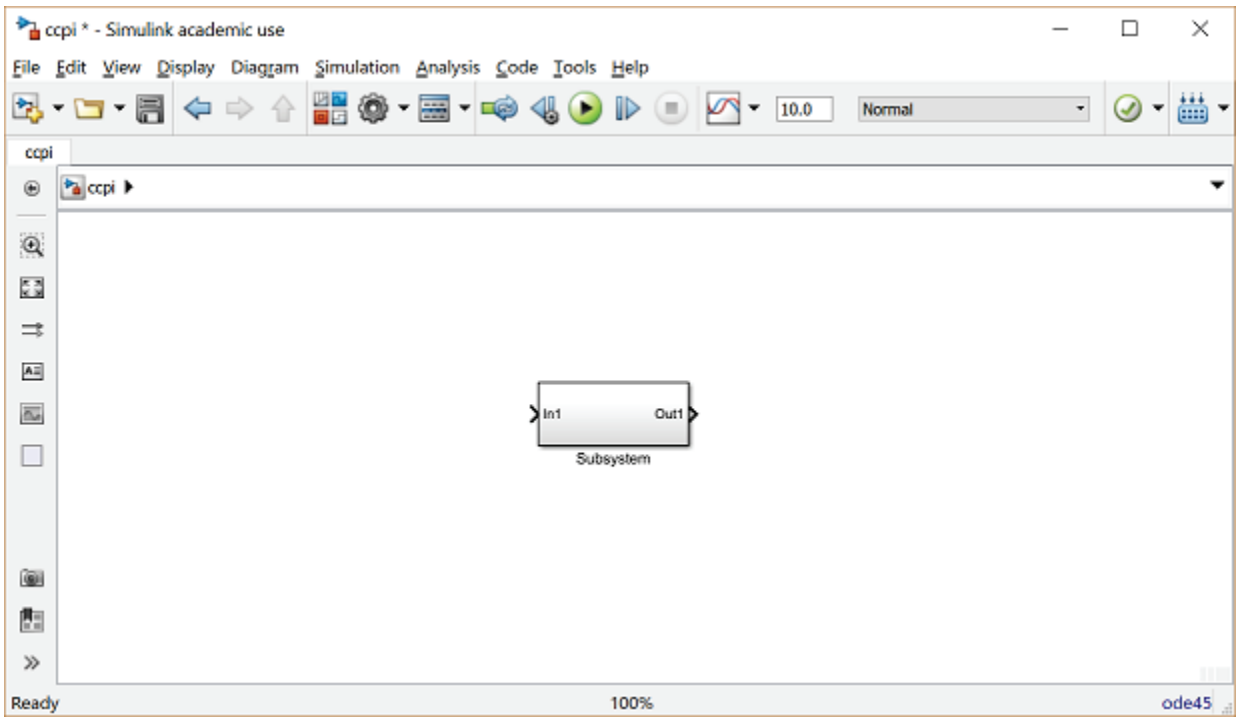
You should obtain the following step response plot



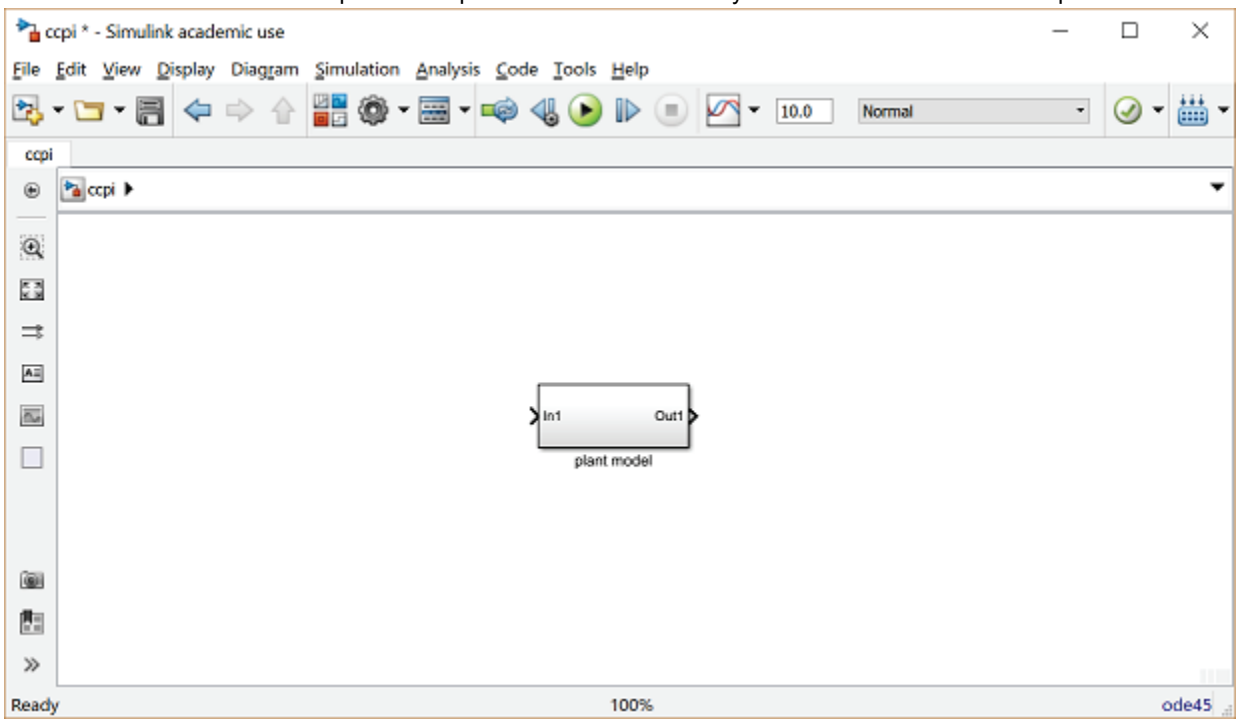
3.4 Closed-loop control

In the previous part of this lab, a PI controller was designed with $K_p = 800$ and $K_i = 40$ to give the desired response. We will implement this in Simulink by first containing the open-loop system from earlier in this page in a Subsystem block.

- Create a new model window.
- Drag a Subsystem block from the Ports & Subsystems library into your new model window.

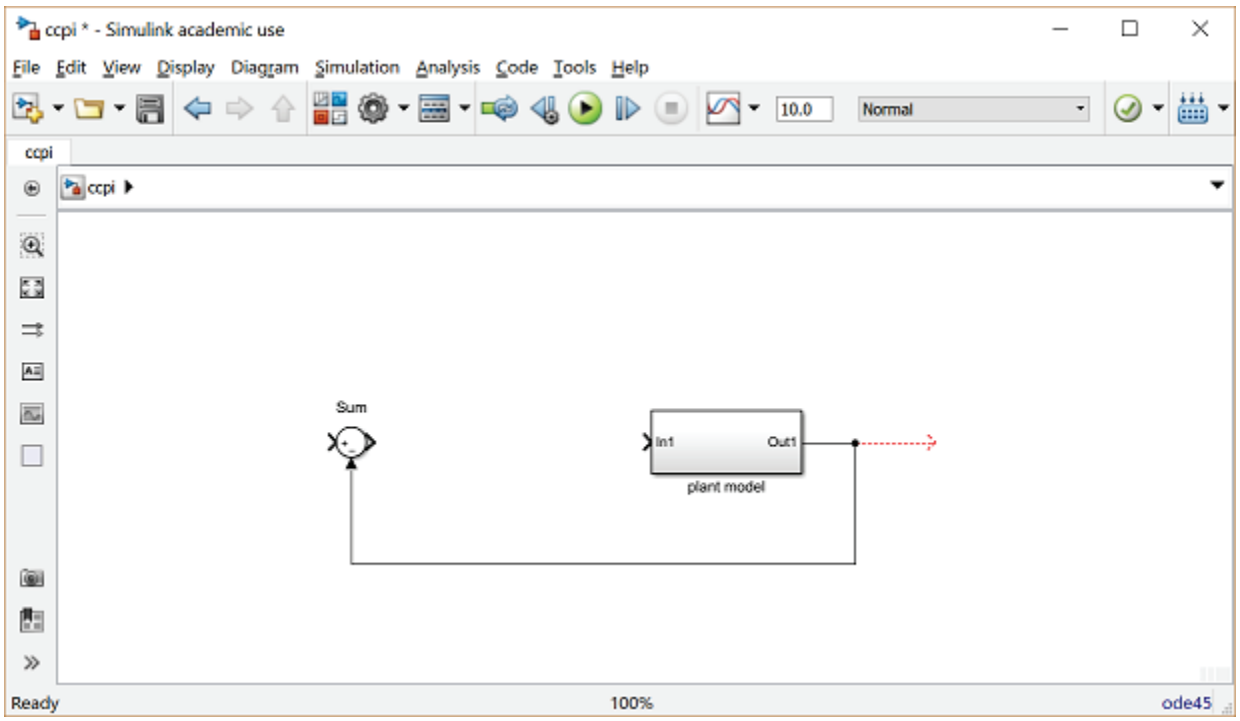


- Double-click on this block. You will see a blank window representing the contents of the subsystem (which is currently empty).
- Open your previously saved model of the cruise control system, ccmodel.slx.
- Select **Select All** from the **Edit** menu (or **Ctrl-A**), and select **Copy** from the **Edit** menu (or **Ctrl-C**).
- Select the blank subsystem window from your new model and select **Paste** from the **Edit** menu (or **Ctrl-V**). You should see your original system in this new subsystem window. Close this window.
- You should now see input and output terminals on the Subsystem block. Name this block "plant model".



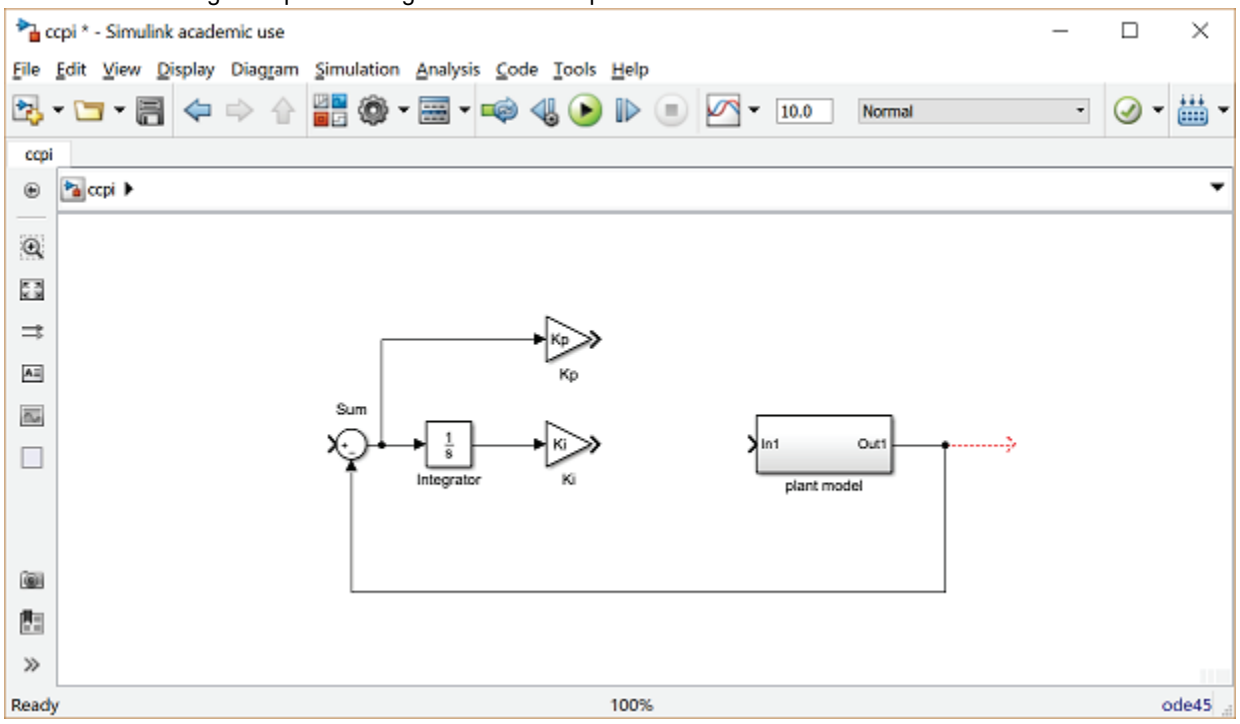
Now, we will build a PI controller around the plant model. First, we will feed back the plant output.

- Draw a line extending from the plant output.
- Insert a Sum block and assign "+-" to it's inputs.
- Tap a line of the output line and draw it to the negative input of the Sum block.



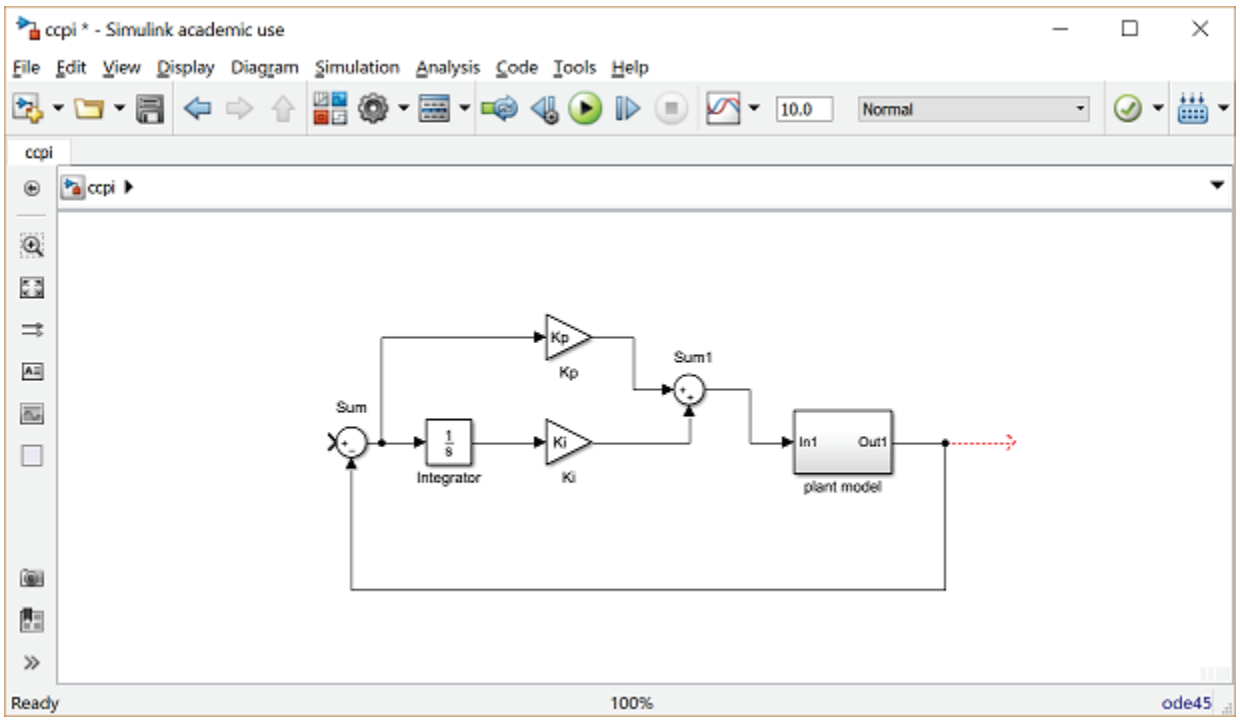
The output of the Sum block will provide the error signal. From this, we will generate proportional and integral components.

- Insert an Integrator block after the Sum block and connect them with a line.
- Insert and connect a Gain block after the Integrator block to provide the integral gain.
- Label this Integrator "Ki" and assign it a value of "Ki".
- Insert a new Gain block and connect it with a line tapped off the output of the Sum block.
- Label this gain "Kp" and assign it a value of "Kp".



Now we will add the proportional and integral components and apply the sum to the plant.

- Insert a Sum block between the Ki block and the plant model and connect the outputs of the two Gain blocks to the Sum block inputs.
- Connect the Sum block output to the input of the plant block.



Finally, we will apply a step input and view the output with a Scope block.

- Attach a Step block to the free input of the feedback Sum block.
- Attach a Scope block to the plant output.
- Double-click the Step block and set the Step Time to "0" and the Final Value to "u". This allows the input magnitude to be changed outside of Simulink.

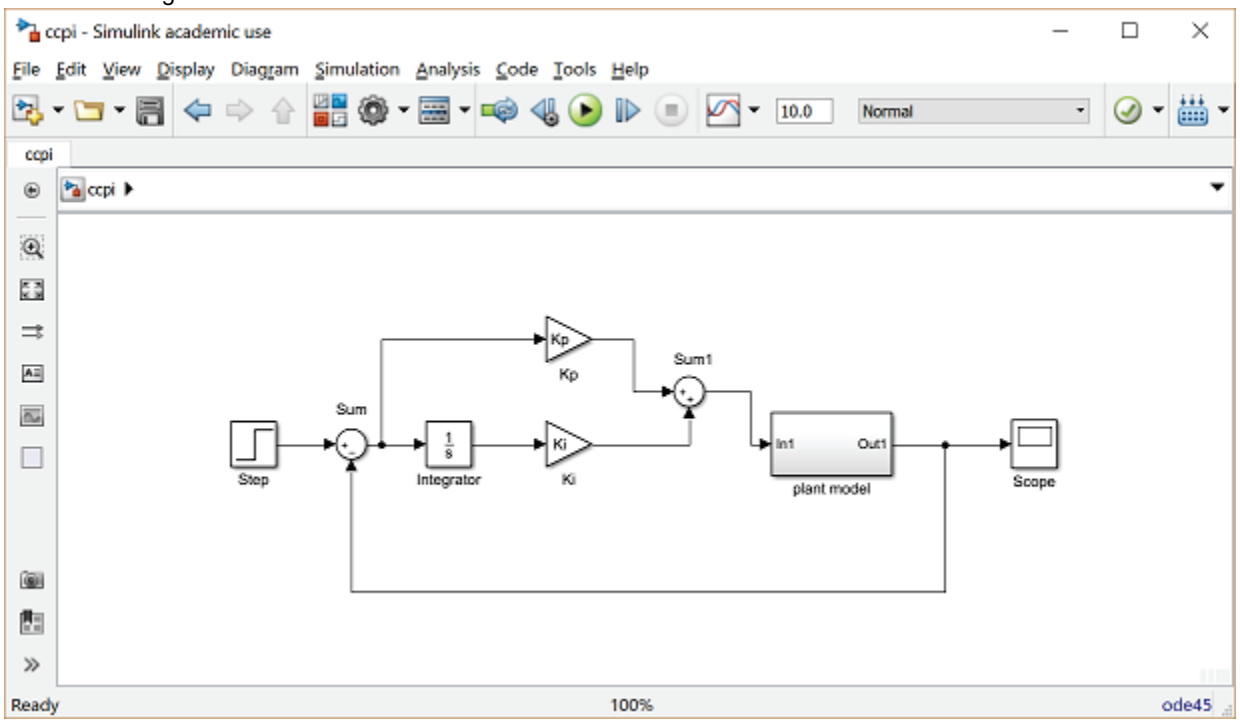


Figure 3.2 Simuling model ccpi.slx

In this example, we constructed a PI controller from fundamental blocks. As an alternative, we could have used a Transfer Function block (from the Continuous library) to implement this in one step, as shown below.

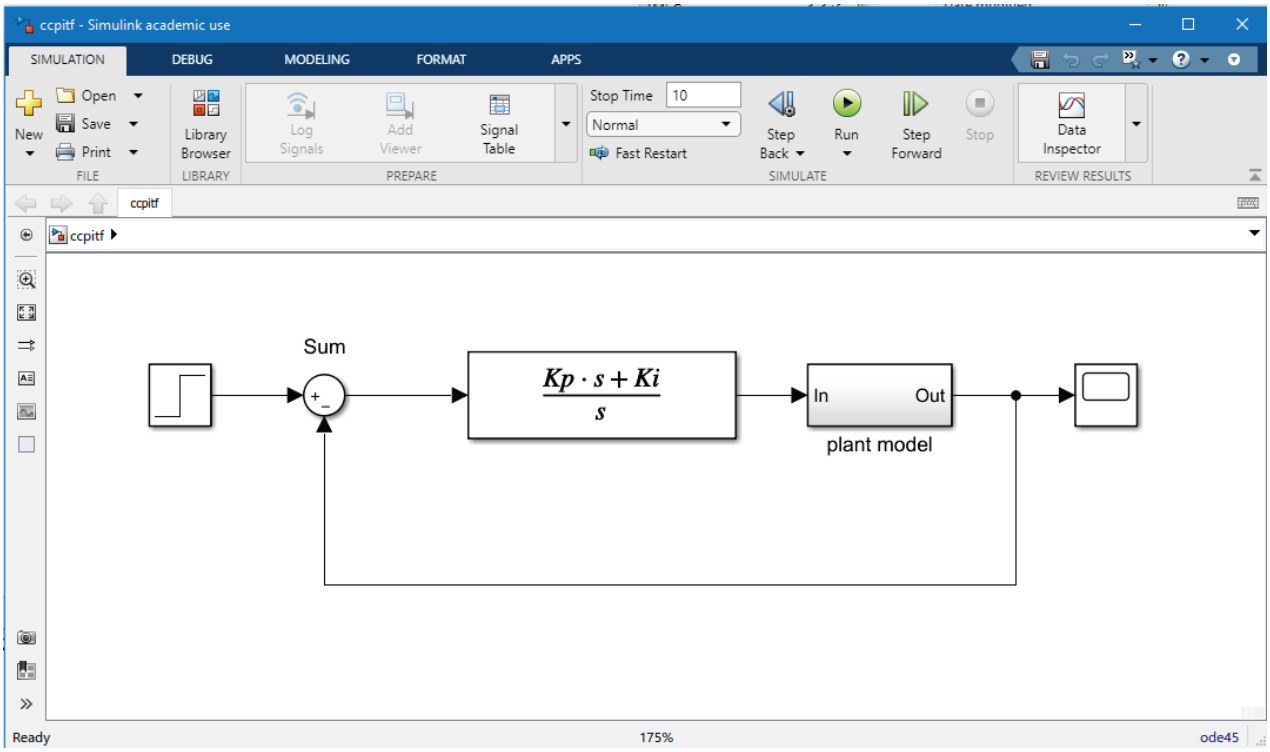


Figure 3.3 Simulink model ccpitf.slx

3.5 Closed-loop response

To simulate this system, first, an appropriate simulation time must be set. Select **Parameters** from the **Simulation** menu and enter "10" in the Stop Time field. The design requirements included a rise time of less than 5 sec, so we simulate for 10 seconds to view the output. The physical parameters must now be set. Run the following commands at the MATLAB prompt:

```
m = 1000;
b = 50;
r = 10;
Kp = 800;
Ki = 40;
```

Run the simulation (hit **Ctrl-T** or select **Run** from the **Simulation** menu).

Questions:

For the PI control system, upload the Simulink files on Blackboard:

Q38(a) The car model file as in Fig. 3.1 (`ccmodel.slx`)
Q38(b) The overall control system model file as in Fig.3.2 (`ccpi.slx`)
Q38(c) The overall control system model file as in Fig. 3.3 (`ccpitf.slx`)

Q39 Upload the step response plot on Scope on **Blackboard**:

END OF VL-1